

## Block 2, Part 4: Server side: Handling data with PHP

---

*Peter Thomson (2017), Dave McIntyre (2019) and Stephen Rice (2021, 2022)*

### 1 Introduction

Over the past two parts of this block, you have been working on client side aspects and focusing on form data validation using both HTML 5 and JavaScript. You have seen how validation on the client side can be implemented so that a form is only submitted to the server if the data is found to be valid.

For the next two parts of Block 2 we will look at the server side and begin to explore some of the techniques and technologies that can be employed to provide good web applications.

There is a wide range of server side programming languages which we will briefly survey before focusing on one example – ‘PHP’ (PHP: Hypertext preprocessor). PHP is a server side language that is especially suited to web development and is used by a high proportion of websites. The practical activities in this part will focus on server side processing of data sent from the forms that you worked on earlier, so that the data can be stored in a data tier and later retrieved and processed for further display in a web page. Once you have mastered these server side aspects you will be able to go on to create a number of basic web applications.

### 2 Learning outcomes


On successfully completing this part of Block 2, you should be able to:

- discuss a range of different server side languages at a high level
- contrast server and client-side languages and operations within a web application
- use the server side facilities that are provided for your TT284 module work and assessment
- recognise and explain some basic PHP code structures to process submitted form data and output HTML content
- create and secure a reusable PHP file and describe benefits and risks of this approach
- explain the risks of creating HTML pages containing untrusted data and describe how malicious content may be handled safely.

### 3 The TT284 server

A workspace has already been created for you on a web server. The TT284 server is provided for your study of TT284 only. Please **do not** upload any other materials that are not directly related to TT284 or which might consume unreasonable resources on the server (memory or CPU). You risk disciplinary action if you do so.

## Activity 1 Access the TT284 server

 Allow 15 minutes for this activity

Before you go further into this part of Block 2, we need to check that you can access the TT284 server. The process is explained in the Guide to TT284 Server Accounts.

The guide is short and concise but contains important information. You should spend some time on the guide, so that you understand how the server is organised and how it should be used.

## 4 Server architecture

In Part 1 we looked at a variety of architectures used to provide a web service infrastructure. This showed that there are a number of different ways of organising what we know as a 'web server'. When researching web servers there are some aspects that can be collected by search spiders and used to build useful statistics, but other aspects, such as the database used, may be hidden behind the server and so statistics may not be readily available.

We have collected information from a variety of sources in mid-2017. Statista is one source that you can access through the OU Library.

The terms that are used here will be explained later as we examine each aspect.

In terms of a server we may use a dedicated hardware box or a set of linked computer boxes, or we might use a virtual web server in the Cloud.

We need an operating system to enable our other software components to run and utilise all aspects of the hardware. Unix (including Linux) runs on about 66% of all servers and Windows about 33%.

We need web server software to handle the incoming and outgoing messages to the internet, and to coordinate other related server software. Apache is the most widely used web server software at close to 50% of market share. Nginx is found on 34% of systems, and Microsoft IIS is at 11%.

Note, while Apache and NGINX are cross platform, IIS runs only on Windows.

We need a server side language, or sometimes more than one language, to handle the business logic and data processing. PHP, the language we shall look at runs about 82% of all websites (but not 82% of all traffic!), ASP.NET about 15% and Java comes third with under 3%.

In addition, we need database software and associated hardware. It isn't so easy to find statistics for the databases used but MySQL and MariaDB are almost certainly the most widely used. There are other products such as SQLite etc. MySQL, MariaDB and SQLite are open source and free to use.

In the next part we will examine the need for a database and explain the structure and function of these databases.

## 5 Server-side languages

Just as in the case of client-side programming, there are several languages that can be used to program the server side. The decision to adopt a language has implications for the other software used in a web application, especially the web server which will be processing the code.

The mainstream candidate languages include:

### PHP

PHP (which stands for PHP: Hypertext Preprocessor) is an open source scripting language and it is the language we shall use in this module. PHP is freely available and well supported online with comprehensive online documentation on the PHP website.

PHP is an open source, general purpose language that can be used for a whole range of applications. It is especially suited to developing web applications because of the ease with which a PHP script can be embedded within HTML, and also can be used to dynamically generate HTML content.

PHP is most commonly used on an Apache server running on a Linux operating system, however it can also be installed on both Windows and macOS systems.

### ASP.NET

Our second candidate is ASP.NET. Microsoft have developed the '.NET framework'. The framework provides a 'Common Language Runtime' which allows code written in any of the Microsoft family of languages to be run on the same machine. There are several '.NET' languages, such as C# (pronounced 'c-sharp'), Visual Basic .NET (VB.NET) and J#, but the technology focused on web development is the .NET version of Active Server Pages (ASP) called 'ASP.NET'.

Until recently using Microsoft technology implied also adopting the Microsoft web server 'Internet Information Services' (IIS). However, in 2015 Microsoft started making ASP.NET available for Linux servers, where it can run on both Apache and Nginx.

On Windows servers, IIS can be used with ASP.NET as well as other non-Microsoft languages, such as PHP.

### Java

Our third candidate is Java. Java is an open source implementation of an object-orientated language originally produced by Sun Microsystems (now part of Oracle). Java is freely available on a wide range of platforms (including Windows, Linux, macOS) and has extensive support documentation, sets of tools and software libraries as well as wide adoption in both industry and academia.

When Java is used to serve dynamic web content it needs a special web server that supports Java, such as Tomcat.

## Python

Python is often used as a server language especially, at the time of writing, for some prominent sites with high traffic levels, such as YouTube and DropBox.

Python is particularly useful for processing large data sets and for handling scientific and statistical data, and for generating visualisations from such data.

It is very easy to install web servers written in Python on your own computer, and use SQLite (which is included with Python) as the database.

## Node.js

Node.js provides a JavaScript run time environment on the server. It is designed to efficiently handle large numbers of requests, and cope with large processing requirements. It can scale to handle these larger processing requirements much more effectively than PHP. It also allows developers to use the same JavaScript language for both server-side and client-side processing.

## 6 Starting with PHP

In this part of Block 2, we will use the open source scripting language PHP and the open source database system called MariaDB. As we mentioned earlier this is already set up for you on the TT284 server.

To run PHP we need a web server that serves content, such as HTML pages. The server must also be able to run PHP scripts. To support this, the web server is extended with a PHP component or 'module'.

A PHP script combines code to be interpreted and processed with HTML code that is included by the server without any further processing.

### Activity 2 Upload the PHP files to the TT284 server

 Allow 15 minutes for this activity

Download the zipped folder of files from the Block 2 Part 4 resources area.

Unzip them into the TT284 module folder on your own computer. Note you cannot run PHP files on your own computer. You must upload them to the TT284 server first. To do this you'll need access to your TT284 Server account.

Instructions on how to access your TT284 server are explained in the Guide to TT284 Server Accounts we encountered in Activity 1.

We recommend that you upload them all now and in the following parts use the editor on the server to edit the files there. If you prefer, you can edit them locally on your machine and upload them for testing as needed, but note that you will have to delete a file from the server before you can upload a fresh copy.

## The typical functions of PHP on the server

PHP is typically used where part or all of the web page needs to be dynamically generated. The data it uses may be retrieved from one or more databases, and might be created for a specific user. Where content is for a specific user, that user first needs to be identified. We can see a common pattern of processes on the server:

1. Generate a welcome page with forms for the user to sign in.
2. Receive the sign in, validate the data, and authenticate the user. Store data in the database. Read data from the database. Process data. Generate an information page – perhaps with details specific to that user. Record the identity of the user for the current session.
3. Receive data back from the user, validate the data. Identify the user from the session. Store data in the database. Read data from the database. Process data. Generate the next information page – perhaps with details specific to that user.
4. And continue to repeat this third series of actions until the user signs out, or the current session expires as a result of lack of user activity.

The process might be as simple as a request for a standard page of information, or a complex form, or for an item of data to be inserted into a page that the user is already viewing. The server repeats this same standard cycle of processes that are outlined above. These supply the required pages or data to the user while also ensuring the security of the server, security of the user and security of the data.

An alternative sequence might be:

1. Generate a welcome page with a choice of tasks and send this to the user.
2. Receive back a request with data on the task to carry out. Generate a new page to respond to that request using data from the database. The full page itself never exists on the server. Only a collection of templates and data in the servers.
3. Repeat this second series of actions.

## 7 Hello World

A PHP file consists of both script to be executed on the server that may generate some HTML code, and HTML code to be directly included in the page sent to the browser.


This is the content of **helloworld.php**.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>PHP Hello World!</title>
</head>
<body>
  <?php
    echo '<h1>PHP Hello World!</h1>
      <p>This script is inside the HTML content and should be run by the server at this point.</p>';
```

```
?>
</body>
</html>
```

Either open it and view in your text editor, or if you already uploaded it, use the editor on the server. Most of this is our normal HTML, but the PHP script is enclosed between the `<?php ?>` tags. The command `echo` is an instruction to insert what is enclosed into the HTML page at that point.

### Activity 3 Upload a file to your server and run it

 Allow 10 minutes for this activity

If you have not already done so, find the file **helloworld.php** – upload it to the server.

Now click on its link to run it.

Note your tt284 server account is located at: <https://oucu.tt284.open.ac.uk/> where 'oucu' is your own personal oucu number, for example, 'abc123'.

Open the Block 2 Upload Page on your server, edit **helloworld.php** and compare the content of the file that you see in the text editor with what is displayed by 'view source' in your browser when the file is run.

Note the differences you see between the source code and the output code and how the `echo` command in PHP has been implemented inside the HTML.

Reveal answer

## 8 Which version of PHP are we using?

There are minor differences between the instructions that can be used with different versions of PHP. The most recent versions also have more security features, so it is useful to know which version has been installed on the server.

PHP has a command to let us know which version is in use. It also provides a lot more useful information for the developer. For security reasons it is best not to make this information available on a production server.

### Activity 4 Open the file **version.php**

 Allow 10 minutes for this activity

Upload the file **version.php** to your server account. If you look at the file itself in your chosen text editor it consists of just one line: `<?php phpinfo(); ?>`

Running the file **version.php** on the server, however, displays lots of information about the features of the php you are using and how it is implemented.

Make a note of the version that is shown.

## 9 Commenting code

Commenting your code means that it is easier to maintain and read, both for yourself and for others who may need to modify your code later. Comments also allow you to leave yourself reminders of why you have implemented something in a particular way.

Commenting also provides a mechanism to help with the debugging process by allowing chunks of code to be selectively made 'invisible'. So you might include code to write out messages about the values being manipulated for debugging purposes, but which you later remove by commenting the message code out when the actual program code is working as you desire.

Just as HTML comments, which are contained within special tags, are ignored by the web browser, the PHP parser recognises and ignores PHP comments in any of the three following styles:

- `//` This is a basic one-line PHP comment
- `/*` This is a C-style PHP comment that can span multiple lines.

Note it must be closed `*/`

- `#` This is a "shell-style" PHP one-line comment.

The 'C-style' comment has the same format as a comment in the C programming language and the 'shell-style' comment has the format of a Linux or Unix comment in a shell script. The names are not important here, just that you have the option to use whichever style you prefer. Most PHP scripts you encounter will use a combination of the first two.

It is important to realise that comments are only useful if they are clearly written. As and when code is updated, comments might also need to reflect the changes. It is a common failure of programmers writing and maintaining code to leave comments which are no longer accurate, and that reduces the maintainability of the code in the future.

## 10 PHP variables and data types

Every programming language uses containers for storing items of data, but not every language creates them in the same way. These containers are called variables, and each has a name that identifies it.

In PHP the name of a variable starts with the `$` character, followed by a letter or the underscore character, but not a number. Where a variable is used only briefly you can use a single letter, such as `$x` or `$y`, but otherwise use meaningful names for all variables such as `$user_age` or `$counter`.

In PHP variable names can only contain alpha-numeric characters (i.e. A – Z, 0 – 9) and underscores (`_`). Note that these characters are case sensitive. `$x` and `$X` are two entirely different variables, but easy to confuse. It is better to keep to lower case variables.

In PHP, like JavaScript, you don't need to state what type of data a variable may refer to; hence we say it is a dynamically typed language. While PHP variables themselves do not have specific type, the data they refer to can be of a number of different types, as follows:

**boolean:** – a value that is true or false.

```
<?php
$dothis = false; // assign the value false to $dothis
$dothis = true; // assign the value true to $dothis

if ( $dothis ) {
    echo "<p>You told me to do this</p>"; // will this line be added to the page?
}
?>
```

**integer:** – whole numbers that may be positive or negative. But beware. A leading zero means an octal number (base 8) 0b precedes binary notation (base 2) and a hexadecimal number starts 0x (base 16).

```
<?php
$mydec = 12345; // decimal
$myneg = -12345; // negative decimal
$myoct = 01234; // starts with 0 so must be a valid octal number
$myhex = 0xFF; // starts with 0x so must be a valid hexadecimal number
$mybin = 0b111000; // starts with 0b so must be a valid binary number
?>
```

**float:** – a floating point number that can be expressed as a decimal or using scientific notation. Beware that a float can have limited precision because of tiny rounding errors, and the inability to represent some fractions as binary numbers. This is very similar to trying to represent the fraction 1/3 as a decimal number. Don't compare floating point numbers directly to see if they are equal, and never multiply small differences by large amounts. The result may not be accurate.

```
<?php
$simple_float = 10.987;
$scientific_float = 1.2e3;
?>
```

**string:** – strings can be created using single quotes or using double quotes. In PHP, single-quoted strings are quite simple to use.

If we want the string represented exactly as written, then the string held inside single quotes means PHP will not try to interpret it. The exception to this is the use of backslash. That allows us for example to include single quotes themselves:

```
<?php
echo 'This is a \'test\' string';
//Output: This is a 'test' string
```



?>

In double-quoted strings, other escape sequences are interpreted as well and any variables will be replaced by their value.

```
<?php
$count = 9;
echo "The count is $count";
//Output: The count is 9
?>
```

Specifying single quotes instead of double quotes for the above example it will be like this:

```
<?php
$count = 9;
echo 'The count is $count';
//Output: The count is $count
?>
```

The most important difference is that variables get replaced by their values in strings within double quotes.


In PHP, strings can also be joined together ('concatenated') using a dot, as in the following example:

```
<?php
$string1 = 'The Cat';
$string2 = 'on the mat';
$string3 = 'got the cream';
$string4 = $string1 . ' ' . $string3;
echo "<p>$string4</p>";
//Output: <p>The Cat got the cream</p>
?>
```

Like JavaScript, PHP will try to guess what you intend when you carry out operations using different data types. An easy way to see what value and data type a variable refers to is using the `var_export()` function:

```
<?php
$mydec = 12345; // decimal
$mystery1 = $mydec + '200'; // add number to string of digits
var_export($mystery1);
//Output: 12545
$mystery2 = $mydec . '200'; // concatenate number to string of digits
var_export($mystery2);
//Output: '12345200'
?>
```

## Activity 5 data-types.php

 Please allow 15 minutes for this activity

Upload the file **data-types.php** to your server space (if you have not already done so) and open it in your browser. You will also need to open it in the editor. Going through this file line by line and comparing what you see in the browser against the text of the PHP demonstrates how PHP uses the script to output HTML to the page.

You should be able to explain each line of output. As before, do make changes to the code and check that you get the expected outputs on the webpage. If there is anything you cannot explain, raise it in the Block 2 Forum.

This file also shows the basics of combining PHP and HTML on the same page. You will use these concepts further.

## 11 PHP arrays and foreach loops

In PHP, arrays can be used to store both lists of values (indexed by number) and dictionaries of keys and values (indexed by string key).

A **list** might be as simple as a shopping list:

- apples
- cabbage
- beans
- eggs
- milk

The order may not be important, but each item in the list can be identified by its position in the list, starting at zero. The number here is the key to the data in the list:

0 apples  
1 cabbage  
2 beans  
3 eggs  
4 milk

In other lists the order is important, and again each item can be identified by its position in the list:

0 put water in pan  
1 put egg in water

- 2 turn on heat
- 3 boil for five minutes.

A **dictionary** uses a unique string of characters as the key to store and to look up the value of the item that is stored, rather like a Python dict or a JavaScript object.

**Table 1 Key and Value pairs using a dictionary**

Key	Value
author	Peter
document	Block 2 Part 4
title	The Server Side (1)
module	TT284

## All arrays in PHP have the same basis

A PHP array is created with 'keys' that can be used to access values. Each key in an array is unique. The key can either be an integer or a string. Any other value will be converted to an integer or string or won't be valid. This type of structure is sometimes known as an 'associative array'.

There are two ways of creating a dictionary-like array for the following table of data:

**Table 2 An array of keys and values**

Key	Value
b	Monday
1	TT284
abcd	The cat

```
<?php
$dictionary1 = array(
    "b" => "Monday",
    "1" => "TT284",
    "abcd" => "The cat"
);
?>
```

There is also an abbreviated notation using [] rather than array():

```
<?php
$dictionary2 = [
    "b" => "Monday",
    "1" => "TT284",
    "abcd" => "The cat"
];
?>
```

You will find both forms in use, and either is fine.

An array can be created without using a key. Here an integer is automatically added as a key, starting from zero. This auto-increments from the previous key. Again, there are two ways of creating a list-like array:

```
<?php
$list1 = array("bread", "butter", "jam", "apples");
$list2 = ["bread", "butter", "jam", "apples"];
?>
```

The result is the same as if we created a table for the following data:

**Table 3 An array with auto-incremented keys**

Key	Value
0	bread
1	butter
2	jam
3	apples

## Extracting data from an array

You can output the value of an array directly:

```
<?php
echo "<p>The first item in my shopping list is {$list1[0]}</p>
    <p>The item abcd in my dictionary is {$dictionary1['abcd']}</p>";
//Output:
The first item in my shopping list is bread
The item abcd in my dictionary is The cat
?>
```

Observe that the curly brackets around the variable names within the double-quoted string are not output. The use of such brackets is optional but can improve code readability and reduce the chance of PHP misinterpreting the variable name.

## Updating a value in an array

To update a value, assign a new value to a key:

```
<?php
$list1[0] = "cake";
$list1[4] = "oranges";

$dictionary1["abcd"] = "The dog";
echo "<p>The first item in my shopping list is {$list1[0]}</p>
    <p>The item abcd in my dictionary is {$dictionary1['abcd']}</p>";
//Output:
The first item in my shopping list is cake
The item abcd in my dictionary is The dog
?>
```

## To remove an item from an array

The unset() function is used to remove items:

```
<?php
unset($list1[3]);
unset($dictionary2["b"]);
?>
```

## To iterate over an array

In PHP, the foreach construct is designed for reading the contents of an array.

Iterate keys and values of a list-like array:

```
<?php
foreach ($list1 as $key => $value) {
    echo "Key: {$key} Value: {$value} <br />";
}
//Output e.g.: Key: 0 Value: cake
}
?>
```

Iterate keys and values of a dictionary-like array:

```
<?php
foreach ($dictionary2 as $key => $value) {
    echo "Key: {$key} Value: {$value} <br />";
}
//Output e.g.: Key: abcd Value: The cat
?>
```

Or iterate just values (e.g. for a list)

```
<?php
foreach ($list1 as $value) {
    echo "Value: {$value} <br />";
}
//Output e.g.: Value: butter
?>
```

## Activity 6 array-types.php

 Allow 30 minutes for this activity

Upload the file **array-types.php** to your server space (if you have not already done so) and open it in your browser. You will also need to open a copy in the editor. Go through this file line by line and compare what you see as output in your browser against the text of the PHP when viewed in your text editor.

You should be able to explain each line of output. As before, do make changes to the code and check that you get the expected outputs on the webpage. If there is anything you cannot explain, raise it in the Block 2 Forum.

Try to work out from the code in the editor how the data will be processed, and what will then be displayed in the browser.

This file demonstrates some of the basics of using arrays in PHP. In the following exercises we'll examine this further.

## 12 Making decisions

Programming is about implementing decisions. A useful means of negotiating programming decisions is by using 'if' statements.

First, we will use 'if' to compare values. Assign some values to the variables to compare:

```
$a = 42;
$b = 37;
$c = "TT284";
$d = "<p>some HTML</p>";
```

Let's work through some comparisons that are available.

We will use a comparison to control an if...then...else statement.

You can think of this as – if this expression evaluates to be 'truthy' i.e. equivalent to true, do this, but if it evaluates to be 'falsey', i.e. equivalent to false, do that.

The comparison `==` is known as a loose comparison, whereas `===` is known as a strict comparison. The strict comparison matches the type as well as the value and is true only if both values being compared are identical. The loose comparison only checks the value and not the type, and so may be true even if the two values being compared are not identical.

Whilst you might expect the strict comparison to be more common, in practice the loose comparison is often more appropriate. So, for example, we may want to compare a floating point number with an integer.

Using loose comparison:

```
<?php
$a = 4; // Use the integer value 4
$b = 4.0; // Use the floating point value 4.0
// Numerically these are the same, but their types are different
if ($a == $b)
    { echo 'Match';}
else
    { echo 'No match';}
//Output: Match
?>
```

Whilst using strict comparison:

```
<?php
$a = 4; // Use the integer value 4
$b = 4.0; // Use the floating point value 4.0
// Numerically these are the same, but their types are different
if ($a === $b)
    { echo 'Match';}
else
    { echo 'No match';}
//Output: No match
?>
```

Clearly the two variables in some sense represent the same value, so depending on your needs you can control the strictness of the comparison.

As well as making decisions based on comparisons between variables, we can also evaluate variables to see if they are 'truthy' or 'falsey'. The set of values equivalent to false is smaller than the set of values equivalent to true – essentially anything that is not 'falsey' is equivalent to true when evaluated in an 'if' condition.

In PHP the most important values considered 'falsey' are:

- integer 0 and float 0.0
- the empty string "" and "0" (because it is equivalent to integer 0)
- an empty array []

- the null value
- and of course, the boolean false.

Anything else is equivalent to true when evaluated in an 'if' or 'while' condition. So we can, for example, decide to carry out a task:

```
<?php
$task = ""; // empty string
if ($task) {
    echo 'There is a task to do';
} else {
    echo 'Nothing to do!';
}
//Output: Nothing to do!
?>
```

To carry out a 'not' operation, to make a true value false and vice versa, we can add the ! operator before the variable or expression:

```
<?php
$valid = false;
if (!$valid) {
    echo '$valid is false';
} else {
    echo '$valid is true';
}
//Output: $valid is false
?>
```

Finally, if we wish to make a decision based on the value of an array element, we can use an 'if' condition, but PHP will output a 'notice' to the HTML page if the element does not exist. To safely check if an array element is missing or falsey, we can use the empty() function, often in combination with the ! operator:

```
<?php
$array = [];
if (!empty($array['id'])) {
    echo 'The array element with key "id" does not exist or is false ';
} else {
    echo 'The array element with key "id" does not exist or is falsey';
}
//Output: The array element with key "id" does not exist or is falsey.
?>
```

## Activity 7 decisions.php

 Allow 15 minutes for this activity



Upload the file **decisions.php** to your server space (if you have not already done so) and open it in your browser. You will also need to open it with the editor. Go through this file line by line and compare what you see in the browser with the text of the PHP when viewed in your text editor.

You should give the variables different values and predict the outcome, then open the file in the browser to verify your expectations.

You should be able to explain each line of output. As before, if there is anything you cannot explain, raise it in the Block 2 Forum.

Try to anticipate from the code how the data will be processed and what the result will be, that will be written as HTML code to the browser for display.

This file demonstrates some of the basics of using 'if', 'else' and 'elseif' in PHP.

Note that to focus on the PHP, we have omitted some of the HTML here and in the next few examples. The resulting HTML page would not validate, but will display the output from the PHP.

## 13 While loops

We have already used the 'foreach' construct to iterate through an array. Another essential way of looping in PHP is the 'while' loop construct. We use this where we expect a test expression to change from true to false, and we want to repeat a process until that change takes place. When programming loops, it is better to use an expression that tests a condition using '>' or '<' or '<=' or '>=', rather than '=='.

### Loops using while

Consider the following PHP script:

```
<?php
$i = 1;
while ($i <= 11) {
    echo "<p>i = {$i}: Do something.</p>";
    $i = $i + 1; // increment the iterator
}
echo "<p>That loop has now ended!</p>";
//Output: "<p>i = 1: Do something.</p>" up to "<p>i = 11: Do something.</p>", followed by <p>That loop has
?>
```

### Activity 8 Testing the concept of 'while' using while.php

 Allow 30 minutes for this activity

Upload the file **while.php** to your server space (if you have not already done so). Open it in a browser to see what it does, then open the file in an editor. Investigate how to change the number of iterations. Again, go through this file line by line and compare what you see in the browser against the text of the PHP.

Consider from the code how the data will be processed and what the result will be, in terms of HTML code interpreted by the browser for display.

If there is anything you cannot explain, raise it in the Block 2 Practical Forum.

The while.php file has shown you the basics of using 'while' to loop in PHP.

**Beware of loops that never end! They waste server resources.**

**Try to think carefully about how it will end before you create a loop.**

## 14 Handling data from forms

PHP is a programming language designed to handle data submitted from HTML forms.

In PHP, data from the HTTP request is made available in 'superglobal' built-in variables. Being superglobal means data are available in any scope. Each superglobal is an array, and as you saw previously, each item in the array has both a key and a value. You can iterate through all the keys in the array by using a 'foreach' construct, or you can look for a specific item using its unique key.

The most important superglobal variables are defined below:

---

<code>\$_SERVER</code>	An array that contains data about the server running the PHP script. <code>\$_SERVER[ 'PHP_SELF' ]</code> can be useful to insert the return URL into a form. If used it should be escaped with <code>htmlspecialchars()</code> as explained at the end of this section.
<code>\$_GET</code>	An array that contains keys and values sent to the server on the end of the URL or by submitting a HTML form using the GET method.
<code>\$_POST</code>	An array that contains keys and values sent to the server by submitting a HTML form using the POST method.
<code>\$_FILES</code>	An array that contains data about files uploaded using a special type of POST request.
<code>\$_COOKIE</code>	An array that contains cookies sent to the server that were previously set by the same server. Cookies will be explored further in Block 3.
<code>\$_SESSION</code>	An array that contains data previously stored on the server in the current session. Sessions will be explored further in Block 3.

---

These superglobals are demonstrated in the file **form-data.php**. This is a general-purpose file and uses a 'foreach' construct to report three kinds of data: That sent by POST, that sent by GET and the FILES.

`$_POST` contains any data sent using HTTP POST from a web page. Usually this is data from a form.

```
<?php
foreach ($_POST as $key => $value) {
    echo '<pre>';
    var_export($key);
    echo ' => ';
    var_export($value);
    echo '</pre>';
}
if (empty($_POST)) {
    echo "<p>No POST data</p>";
}
?>
```

## Activity 9 Compare data submitted by the form with data received by the server using POST

 Allow 15 minutes for this activity

From your Part 4 files, upload **form-data.php** to your server space (if you have not already done so).

Click form-data.html to open it in your browser.

Submit data using the text, select, radio and checkbox inputs. Look at what is shown under the checking POST section.

All of these examples send their data using POST, so the data will appear in the first section. Notice the URL of the response.

## Activity 10 Compare the URL with data received by the server using GET

 Allow 15 minutes for this activity

Now edit **form-data.php** and look for the first form. Change its method from POST to GET.

You can also change other forms if you wish.

Open the form in the browser again and submit some data for that first form (or any you changed to GET). Notice that the data now appears in the GET section, but also notice the URL which now has the data appended in plain sight – not ideal for sending your password, as URLs are often logged by web servers or recorded in web browser history!

There is an important security point here: POST is better than GET for data such as passwords as it sends data in the HTTP request body rather than in the URL. However, POST alone is not enough as data can still be eavesdropped as it passes through the network unless the request is encrypted using the HTTPS protocol.

Finally `$_FILES` contains data about file uploads from a web page.

You will see this data if you use the last form to choose and submit a file for upload. This section illustrates the range of data available about the upload.

## Activity 11 Testing a file upload to the server

 Allow 15 minutes for this activity

Use the file upload field at the bottom of **form-data.php** to upload a small file to the server. Look at what is shown under the FILES section.

Can you match the output to the code above?

Note that in this example we are only checking the file data. The uploaded file is discarded.

# 15 Handling malicious content

## The risk of malicious content

There is always a risk that data sent to the server could contain malicious content. Handling this correctly is an important security measure that web application developers must take.

For text content that will be displayed on a web page, the most common risk is that HTML tags are included in the text content, and specifically HTML script tags that contain malicious JavaScript. This content may be stored in the web server database and used later to create a page. The JavaScript would then run when any user views a page containing that content. This user could be a member of the public thinking they are viewing a trusted website, allowing the JavaScript to steal their data, or could be a user logged into a web application, allowing the JavaScript to take actions on their behalf. This risk is a form of code injection attack known as 'cross site scripting' or XSS .

We could try to remove any malicious content when we receive it, but this can be unreliable and may result in other complications as some data, such as passwords, shouldn't be altered. It is a much better solution to assume that any data could be malicious, and to use methods which we will look at later of storing this safely in the database, and then to ensure that everything we output to a web page is made safe for the user.

When we output any content from our database, the first step we must take is to replace any characters that might be used to embed content, with HTML entities that display these characters instead.

When using PHP, the `htmlspecialchars()` function does the replacement for us.

Some example replacements:

Character	Replacement
& (ampersand)	<code>&amp;amp;</code>
< (less than)	<code>&amp;lt;</code>
> (greater than)	<code>&amp;gt;</code>

This ensures that content such as `<script>alert("attacked!")</script>` is displayed by the browser as text, not interpreted as HTML with the code inside the `<script>` tags being evaluated as JavaScript. The **form-data.php** example in earlier activities intentionally does not protect the user from such content, so you can try to attack this if you wish.

## 16 Reusing PHP code – include and require

It is good programming practice to create a function, or to have a separate file for any code that might be needed to run on more than one occasion.

In such cases instead of repeating these sections of code we need to write it only once, and then insert it into the script that needs it dynamically. Then, should it need to be modified we know that we only need to make the required changes in one place. This technique can also be used to break functionality into more manageable pieces.

In PHP, we can use 'include' or 'require'. Using 'include' will include the code if the file can be found, but omit it otherwise and continue. We are using 'require' so that if the script being inserted is missing, PHP will stop execution and report an error.

### The risk of malicious execution

The PHP files we insert using 'require' can be executed by a malicious user if they know the URL of the file. While the names of these files may not be visible, there is no means to prevent a malicious user from guessing the name of a file and requesting it directly, so we need a way to prevent those files from doing anything unless they are inserted by our own code.

That is important because if a user did request the other files, they might get misleading or partial data, or it might even be possible for them to use them to access data we did not intend.

To achieve that we will define a 'flag' to indicate that it is safe for the files to run, set it in PHP files that should be requested by URL, and look for it in files that should only be executed using require or include. If this flag is not set, the code will refuse to execute.

This demonstration takes three sections of a PHP file that will be used many times; the head (require-head.php) the foot (require-foot.php) of an HTML document and the script to collect data from a form (require-form.php), and places them in separate PHP files. You can find these files in the Block 2 Part 4 download. We will look at one of those files in a moment, but first we will look at the file which draws them all together:

## **require.php.**

This file will be requested by URL so it sets a 'SAFE\_TO\_RUN' flag using the `define()` function to create a PHP constant. It also creates some variables with data (the page title and footer text as an example), which will then be available to the required files. By supplying this data from this file, the same 'head' and 'foot' files can be used with different titles and footers when called from other files making them more useful. Part of this file is shown below:

```
<?php
// Define a constant to permit each file we "require" to execute
// For security, required PHP files should "die" if SAFE_TO_RUN is not defined
define('SAFE_TO_RUN', true);
// Set variables used by required files
$mytitle = 'My page title'; // text to be used by head.php as title
$myfooter = 'My page footer'; // text to be used by foot.php as footer
$url = $_SERVER["PHP_SELF"]; // URL of this page for forms to POST to
// Output the page head
require 'require-head.php';
?>
```

The **require-head.php** file below starts by testing if 'SAFE\_TO\_RUN' has been defined. If not, it 'dies' displaying a message that the file cannot be executed directly.

The message is aimed to be helpful but without revealing details of what this file really does, as a malicious user will use our error messages to better understand and attack our system.

## **require-head.php**


This file contains the security check followed by the start of the HTML page, making use of the title variable created in **require.php**:

```
// For security, required PHP files should "die" if SAFE_TO_RUN is not defined
if (!defined('SAFE_TO_RUN')) {
    // Prevent direct execution - show a warning instead
    die(basename(__FILE__) . ' cannot be executed directly!');
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title><?php echo $mytitle ?></title>
</head>
<body>
```

The files **require-foot.php** and **require-form.php** are of a similar structure. Note that whilst these files are quite simple, that is typically not the case. In a real application the head file is likely to contain the navigation and placing it there ensures consistency across pages and allows a single place where the navigation can be updated. Similarly, the foot is likely to have quite extensive linking to terms and conditions and so on.

With these simple files it can seem more effort to divide them up than is gained, but with a more complex head and foot on a site with a significant number of pages the technique is invaluable.

## Activity 12 Make sure all these files are on the server

 Allow 30 minutes for this activity

The PHP file that 'requires' the files **require-head.php**, **require-foot.php** and **require-form.php** is **require.php**. All four files should now be uploaded to your server account (if you have not already done so).

Click **require.php** to open it in your browser.

View source in your browser and check that the title has been added in the header.

Now open each of these four files in your text editor to examine the content, and see where each part of the created web page has come from and how the page was put together.

Now try editing the content of the required files and upload them to the server to see how this becomes part of the file delivered to the browser.

Note that we have named the security flag 'SAFE\_TO\_RUN', but it could be any name of your choice as long as your files check for this constant in every included or required page at the start. This must be at the start of each file.

## 17 Display data as a table

After we have read form or database data we may want to send it to the user in the form of a HTML table. This requires PHP to output the HTML table tags in the right places between the data items.

This is the main content of **html-table.php**

```
<table border="1">
  <caption>fruits</caption>
  <tr>
    <th>key</th>
    <th>value</th>
  </tr>
  <?php foreach ($fruits as $key => $value) { ?>
    <tr>
      <td><?php echo htmlspecialchars($key) ?></td>
```

```
<td><?php echo htmlspecialchars($value) ?></td>
</tr>
<?php } ?>
</table>
```

Building a table using a loop like this is a very common requirement which you are likely to use often.

#### fruits

key	value
f	apple
e	banana
a	lemon
d	orange
b	plum
c	tomato

### Activity 13 Run `html-table.php` on the server (it requires other files that you have already uploaded)

 Allow 30 minutes for this activity

Upload **html-table.php** to your server space and open it in your browser to observe the results of each section. Open it using the editor so you can see what each section of PHP generates. Note that this file requires you to upload **require-head.php** and **require-foot.php** from the earlier activity.

Can you match the code to the way the table is created?

Now create your own array of data and output the content as a table.

Discuss your code and any problems that you have in the Block 2 Forum.

## 18 Summary

In this part, we have introduced the TT284 PHP server and examined how PHP scripts may be constructed to process form data, make decisions and safely output HTML content.

## Further reading

There is extensive documentation for PHP available online:



- The PHP Manual is the main, very comprehensive, PHP reference, with many examples, although many of the user comments at the bottom of each page are out of date.
- The w3schools PHP Tutorial is also a useful starting point.

There are several online checker sites that allow you to run code online and identify errors, including:

- PHP code syntax check
- PHP Sandbox

Most problems that you may encounter have been discussed by other people. Stack Overflow can be very useful but beware of out of date content.

Do look to see what others recommend in the TT284 Block 2 Forum, and make your own recommendations if you have previously studied PHP.

## **Where next?**

Block 2 Part 5 looks at how we make use of a database with PHP as we create web pages and interact with the user.

---